GIT

Master 2 - IGM

Thibault SAUSSAC
saussact@gmail.com
@ThibaultSaussac

Agenda

Présentation de GIT

- A propos version control
- Pourquoi GIT ?

Git basics

- Comment ca marche ?
- Les commandes de bases
- Git branching
- Merge conflict
- Rebasing

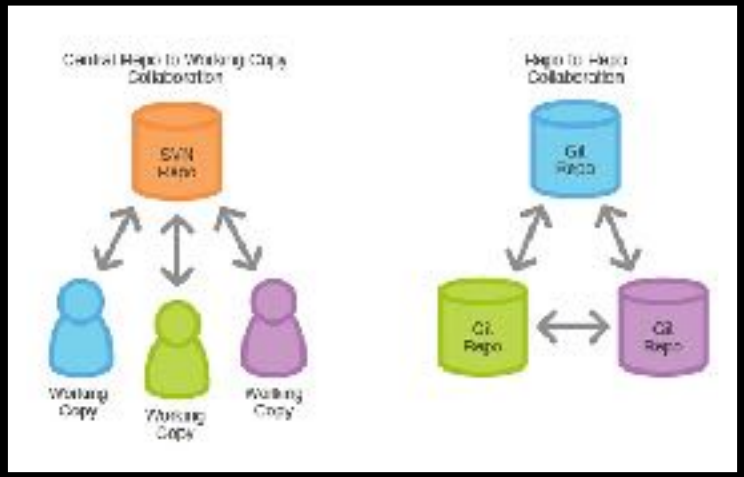
A propos version control

- Source code tracking and backup
  - Version control software records text files changes over time
  - Historique des modifications sauvegardé.
    - Récupération d'ancienne version
    - Comparer les changements
      - En cas d'erreur, c'est facile de faire un 'revert'

- Collaboration
  - Permet les merge de toutes les modifications dans une version commune

    ➔ Tout le monde peut travailler sur tous les fichiers en meme temps.

- Les deux types de VCS
  - Centralized: CVS, SVN
  - Distributed : Bazaar, Git



Pourquoi Git?

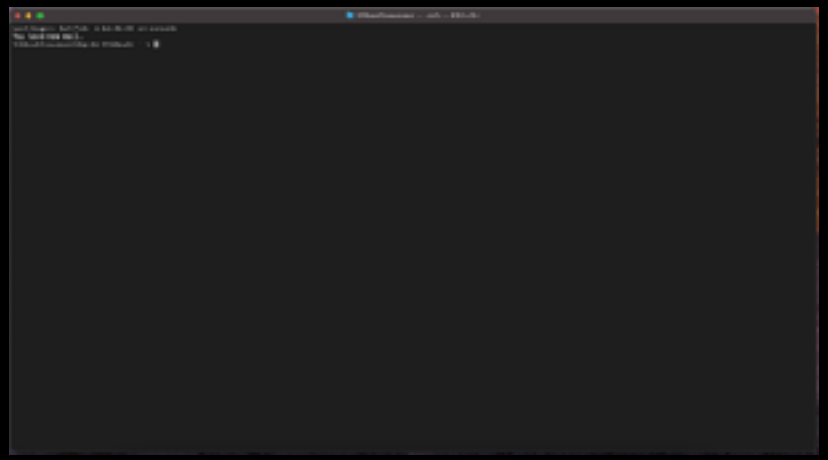
- Git est un VCS distribué
  - Travailler en local sur une copie complète avec l'historique complet du projet
    - ➔ Toute les opérations sont faites en locale : rapide & hors ligne

- Git est l'étoile montante des VCS. Quelques grands projets qui l'utilise :
  - Linux Kernel
  - Fedora
  - Android
  - VLC
  - Twitter
  - Orange SA

Command line

- 2 façons d'utiliser Git
  - Command line
  - Les GUIs
- The command line
  - Seul moyen de lancer toutes les commandes git
  - Si tu maitrise bien les lignes de commandes le passage aux GUI sera simple.
- GUI sont une question de gout personnel
  - Dans tous les cas la command-line tool est disponible.

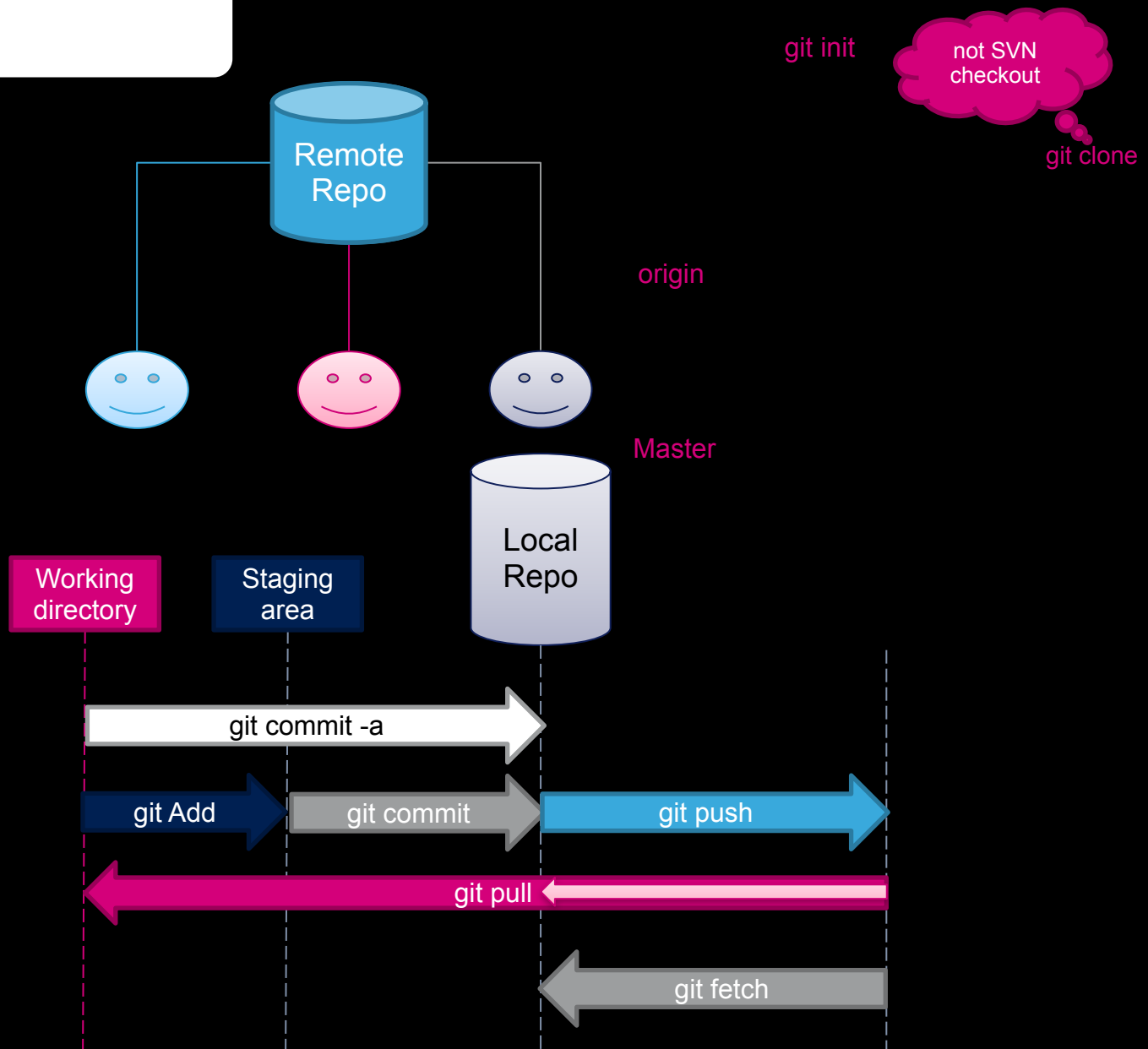
we will train on TP =)



How it works

Basics:
- Origin: default remote
- Master : default branch

Git commands
- git init
- git clone
- git add
- git commit
- git push
- git pull
- git fetch



Git essential commands

Basics

Use git help [command] if you're stuck
- **master : default branch**
- **origin: default remote**
- **HEAD: current point**

Create Repository

Create a new local repo
```
$ git init
```
Clone existing repository
```
$ git clone < repo_url >
```

Local changes

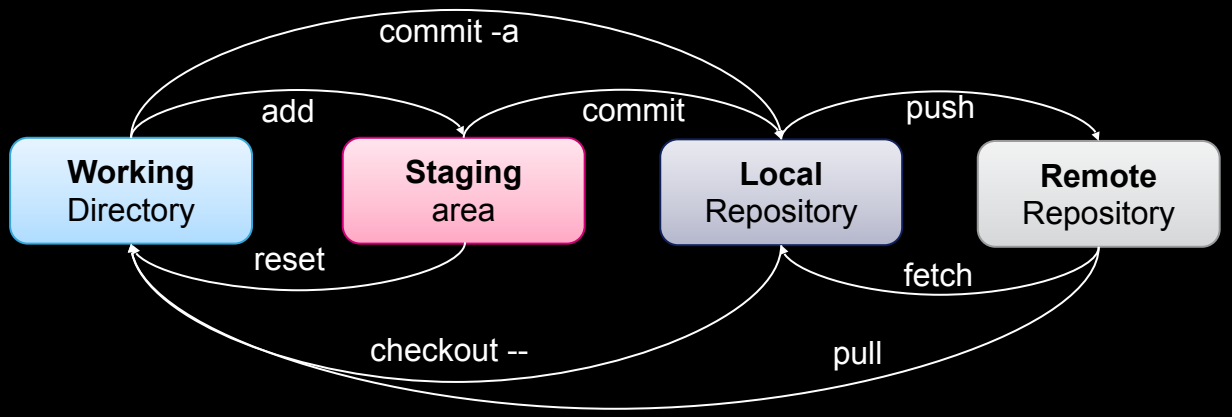
Add files to tracked / staged
```
$ git add file1 file2
$ git add .
```
Commit
```
$ git commit -m "commit msg"
$ git commit -am "commit msg"
```
List changed / new files on local repo
```
$ git status
```
List changes on tracked files
```
$ git diff
```
Show entire history
```
$ git log
```
Show commit content
```
$ git show $id
```

Revert

Unmodifying a modified File
```
$ git checkout -- [file]
```
Unstage a file
```
$ git reset [file]
```
Change last commit
```
$ git commit --amend
```

Synchronize

Push local changes to remote
```
$ git push <remote> <branch>
```
Get the latest changes (no merge)
```
$ git fetch <remote>
```
Fetch and merge last changes
```
$ git pull <remote> <branch>
```



Install and configure Git

- Install Git
  - Ubuntu: sudo apt-get install Git
  - Windows: http://git-scm.com/download/win

- First-time Git setup
  - Configuration is done only once, You can change it at any time.
  - git config command allows you to get and set configuration variables
  - Variables are stored in gitconfig file
  - gitconfig file can be stored in three different places
    - Linux:
      - Configuration for every user in system : **/etc/gitconfig**
      - Configuration specific to user : **~/.gitconfig or ~/.config/git/config**
      - Configuration specific to project **:** **.git/config** (in the git directory)
    - Windows:
      - Configuration for every user in system : **C:\Documents and settings\All users\Application Data\git\config**
      - Configuration specific to user : **C:\Users\$USER\.gitconfig**
      - Configuration specific to project **:** **.git/config** (in the git directory)

Git essential commands

Basics

Use git help [command] if you're stuck
- **master : default branch**
- **origin: default remote**
- **HEAD: current point**

Create Repository

Create a new local repo
```
$ git init
```
Clone existing repository
```
$ git clone < repo_url >
```

Local changes

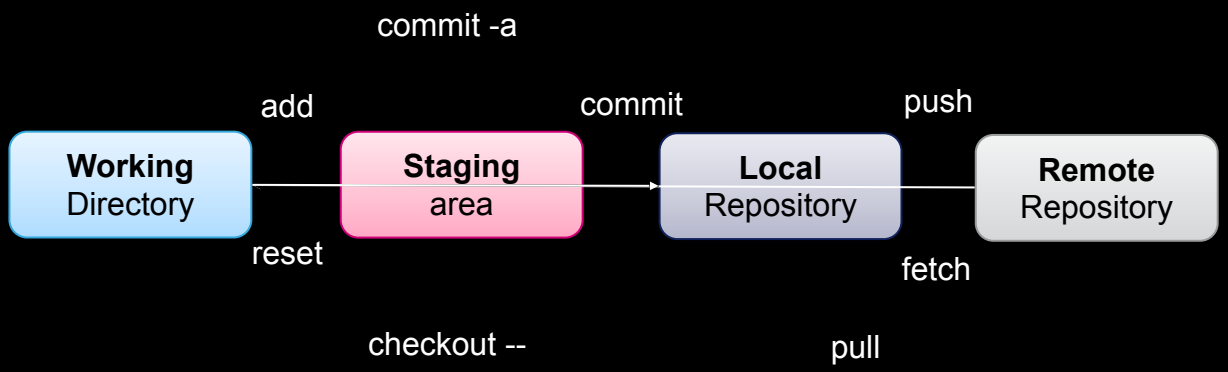
Add files to tracked / staged
```
$ git add file1 file2
$ git add .
```
List changed / new files on local repo
```
$ git status
```
List changes on tracked files
```
$ git diff
```
Commit
```
$ git commit -m "commit msg"
$ git commit -am "commit msg"
```
Show commit content
```
$ git show $id
```
Show entire history
```
$ git log
```

Revert

Unmodifying a modified File
```
$ git checkout -- [file]
```
Unstage a file
```
$ git reset [file]
```
Change last commit
```
$ git commit --amend
```

Synchronize

Push local changes to remote
```
$ git push <remote> <branch>
```
Get the latest changes (no merge)
```
$ git fetch <remote>
```
Fetch and merge last changes
```
$ git pull <remote> <branch>
```
Add a new remote
```
$ git remote add <name> <url>
```
List remote's name and URL
```
$ git remote -v
```

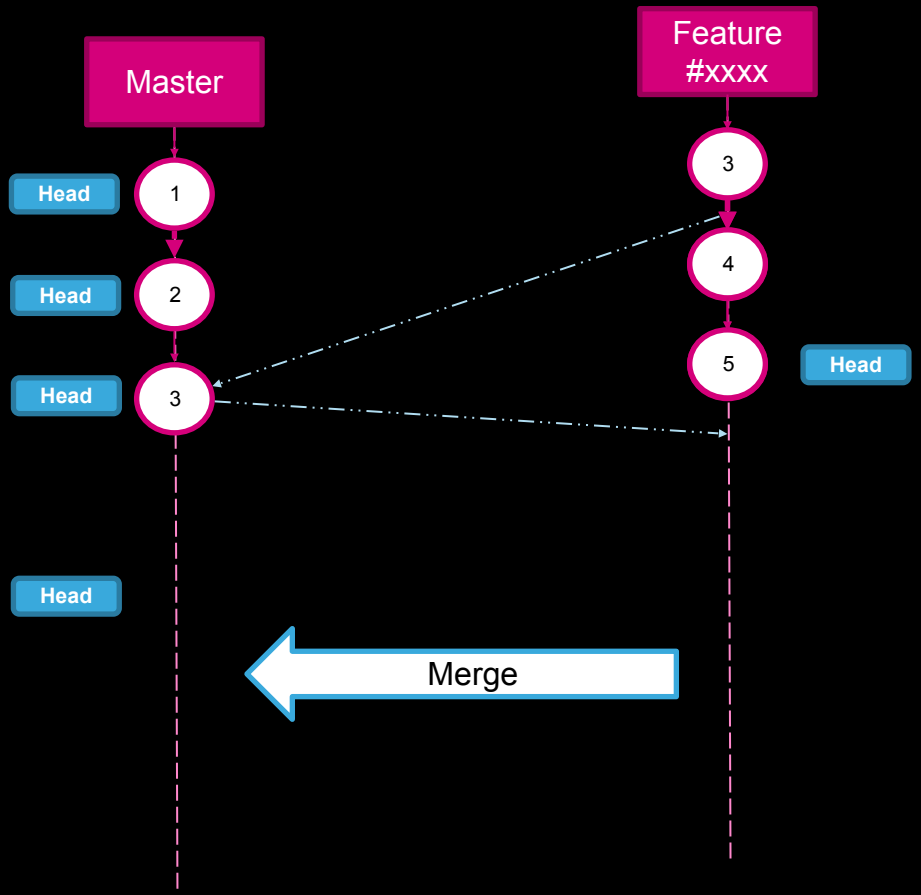


Git commands
- git branch
- git checkout
- git merge

≠ SVN checkout

Git branching

- Les branches : Diverger de master et y revenir « sans » impact.
- Mon avis: Les branches c'est mal ! —> Trunk Base development
  - Gérer les branches c'est complexe… gérer les merges encore plus !
  - Comment gérer les refactoring?
  - https://lhauspie.wordpress.com/2018/05/04/feature-branching-is-evil/



branch

checkout

checkout

merge

Git essential commands

Basics

Use git help [command] if you're stuck
master : default branch
origin: default remote
HEAD: current point

Create Repository

Create a new local repo
$ git init
Clone existing repository
$ git clone < repo_url >

Local changes

Add files to tracked / staged
$ git add file1 file2
$ git add .
List changed / new files on local repo
$ git status
List changes on tracked files
$ git diff
Commit
$ git commit -m "commit msg"
$ git commit -am "commit msg"
Show commit content
$ git show $id
Show entire history
$ git log

Branches

Create branch named <branch>
$ git branch <branch>
Switch to a <branch>
$ git checkout <branch>
Create and checkout a new branch
$ git checkout -b <branchName>
List all branches
$ git branch -a
Delete a branch
$ git branch -D <branchTodelete>
Delete a remote branch
$ git push origin --delete <branch>

Merge

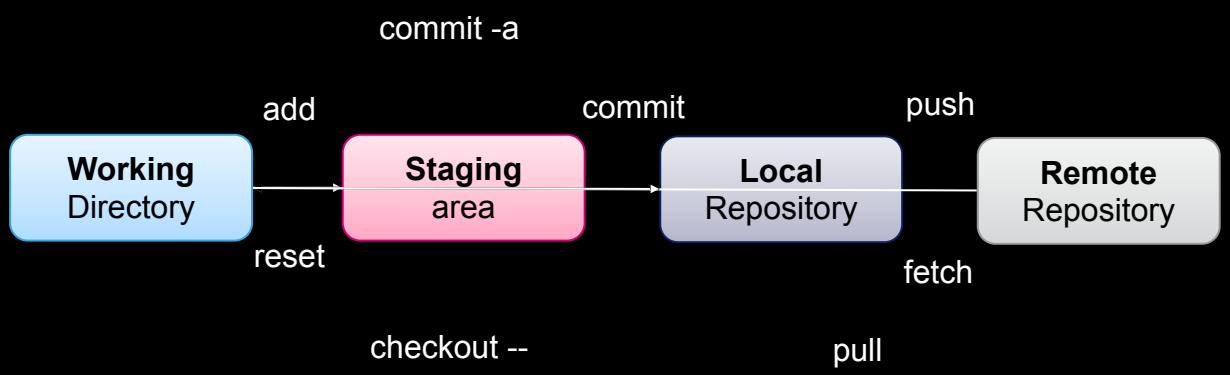
Merge <branch> into current branch
$ git merge <branch>

Revert

Unmodifying a modified File
$ git checkout -- [file]
Unstage a file
$ git reset [file]
Change last commit messages
$ git commit --amend

Synchronize

Push local changes to remote
$ git push <remote> <branch>
Get the latest changes (no merge)
$ git fetch <remote>
Fetch and merge last changes
$ git pull <remote>
Add a new remote
$ git remote add <name> <url>
List remote's name and URL
$ git remote -v



LAB 2 – Branching

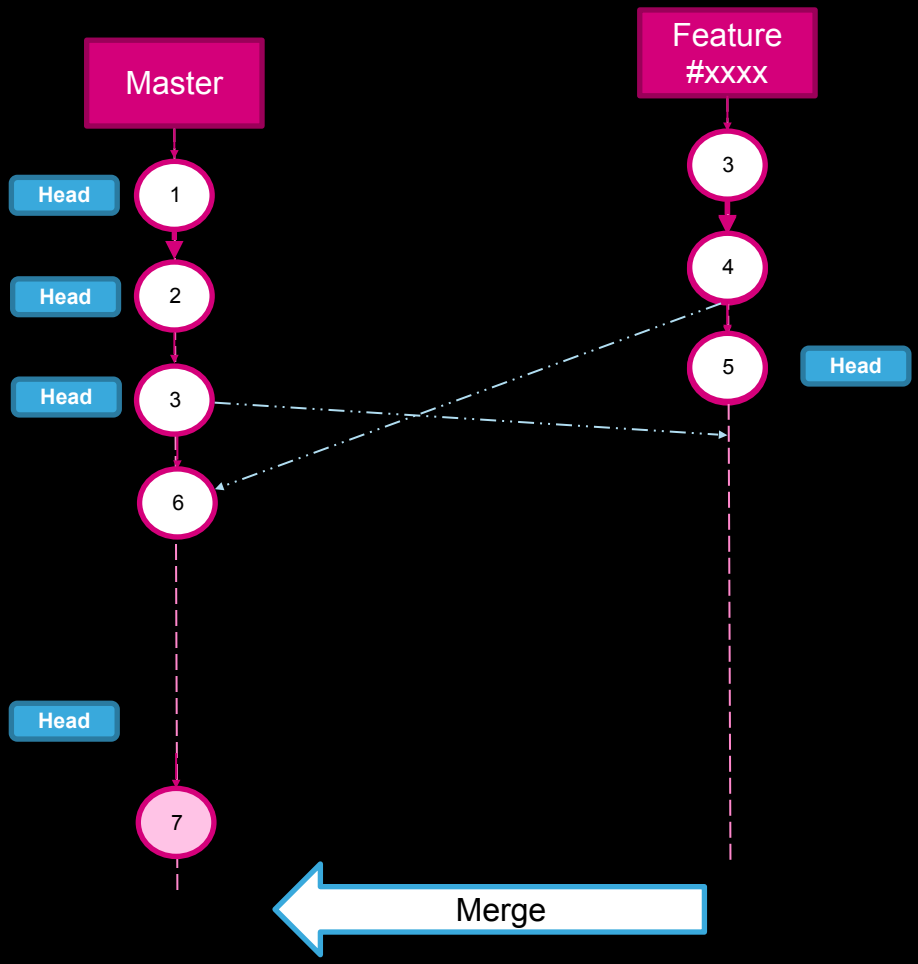
Merge conflict resolution

- Merge conflict: 2 personnes changent le meme fichier
  - Si une personne supprime une ligne et qu'une autre la modify, Git ne saura pas corriger ca.

- git diff pour voir les merge conflits
- git status conflicting files
  - Conflicting files are listed as unmerged
- Fix merge conflicts manuellement dans chaque fichier
  - Local changes between <<<<< HEAD and =======
  - Remote changes ======= and >>>>> branchName
- Valider la resolution en faisant git add <file>
- Pour annuler un merge
  - Git merge --abort

Vous pouvez aussi utiliser pas mal de GUI, qui pourrons vous aider :

- git config --global merge.tool kdiff3
- git mergetool



Git additional commands

Tagging

Patching

Debugging

Cherry pick

Rebase

Resolve merge conflicts

To view merge conflicts

```
$ git diff
```

To discard conflicting patch

```
$ git reset --hard
$ git rebase --skip
```

After resolving conflig, merge with

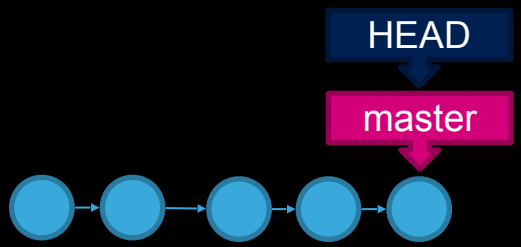
```
$ git add [conflict_file]
$ git rebase --continue
```

Others

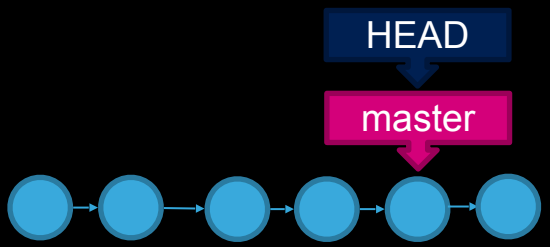
Stash

Reset & Revert

- Reset (BEFORE push)
  - Moves branch pointer sur un commit specific
    - git reset HEAD~1
  - Annuler mes modifications
    - git reset --soft : les modifications sont gardés
    - git reset --hard: modification sont PERDU ⚠



- Revert
  - Annuler un commit en créant son commit opposé.
  - C'est la bonne façon d'annuler un commit, cela ne ré-écrit pas l'historique.



reset vs revert

- git reset est utiliser pour annuler des modifications d'une private branch
- git revert est utiliser pour annuler des modifications d'une public branch

Git command
- git reflog

Git reflog

- Avec Git on ne perd rien !
  - Toutes les actions sont stockés dans le référentiel (commit, pull, push,)
- Reflog affiche la liste de commit à partir de HEAD
  - Exemple de trace ➔

- Reflog expiration
  - Default = 90 days

Git additional commands

Tagging

Patching

Debugging

Cherry pick

Rebase

Rebase master content into current br

```
$ git rebase master
```

Interactively rebase current branch

```
$ git rebase –i  <branch>
```

Resolve merge conflicts

To view merge conflicts

```
$ git diff
```

To discard conflicting patch

```
$ git reset --hard
$ git rebase --skip
```

After resolving conflig, merge with

```
$ git add [conflict_file]
$ git rebase --continue
```

Others

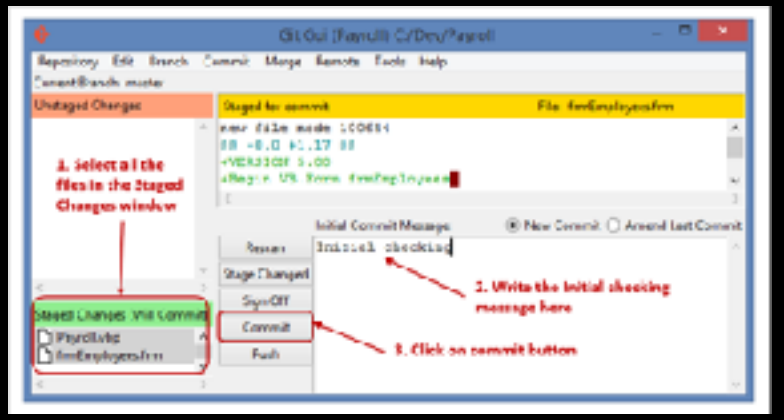
Stash

Git commands
- git gui
- gitk

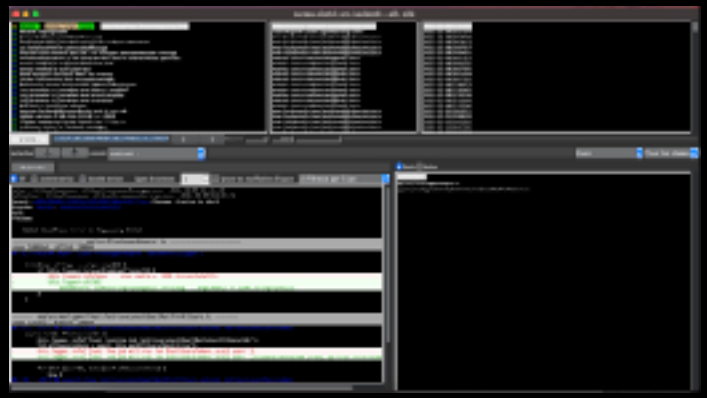
GUI tools

git gui & gitk

- git gui (for committing)
  - Apporter des modification au repo
    - Faire des nouveau commit,
    - Amender un comit,
    - Créer des branches,
    - Faire des local merges
    - Pushing vers le repo



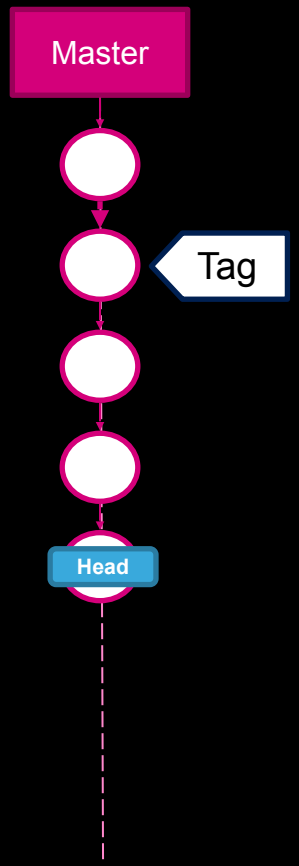
- gitk (for browsing)
  - Affiches les modifications
    - Visualiser le graph des commit,
    - Affiche les commits messages/infos
    - Affiche le diff de chaque commit.



Git commands
- git tag -a v1.0 – m “msg”
- git format patch
- git apply
- git am

Tagging & patching

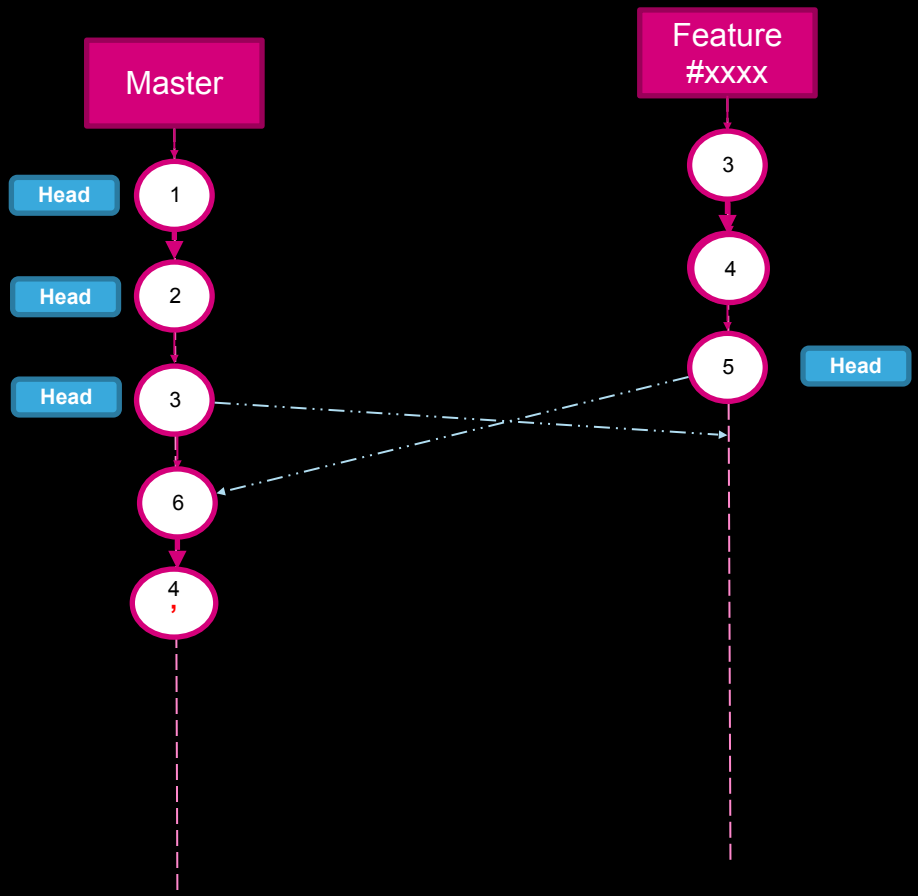
- Tag
  - Permet d’identifier un commit avec un label « human readable »
  - Très utilisé pour marquer les releases
    - Create a tag : git tag –a v1,0 –m "version 1,0 stable”
    - Push a tag: git push <remote> tagName
    - Push all tags: git push <remote> --tags
    - Checkout tag: git checkout tagName
- Patching
  - Permet de partager des changement qui ne sont pas pret a etre pushed
  - Patch c’est juste une concatenation de chaque commits
    - Method 1 (no commit)
      - Create patch : git diff <from-commit> <to-commit> > output-patch-file
      - Apply patch : git apply output-patch-file (no commit)
    - Method 2 (with commit, more formal and keeps authors name)
      - Create for last 2 commits: git format-patch -2
      - git am <name_of_patch_file>



Git commands
- git cherry-pick “commit ID”

Cherry pick

- Je veux récupérer un commit d’une autre branche pour le back porter sur la mienne.
  - git cherry-pick “commit ID”
  - Ex: git cherry-pick af24a94 (commit 4)



Git uses commands
- git stash
- git stash pop

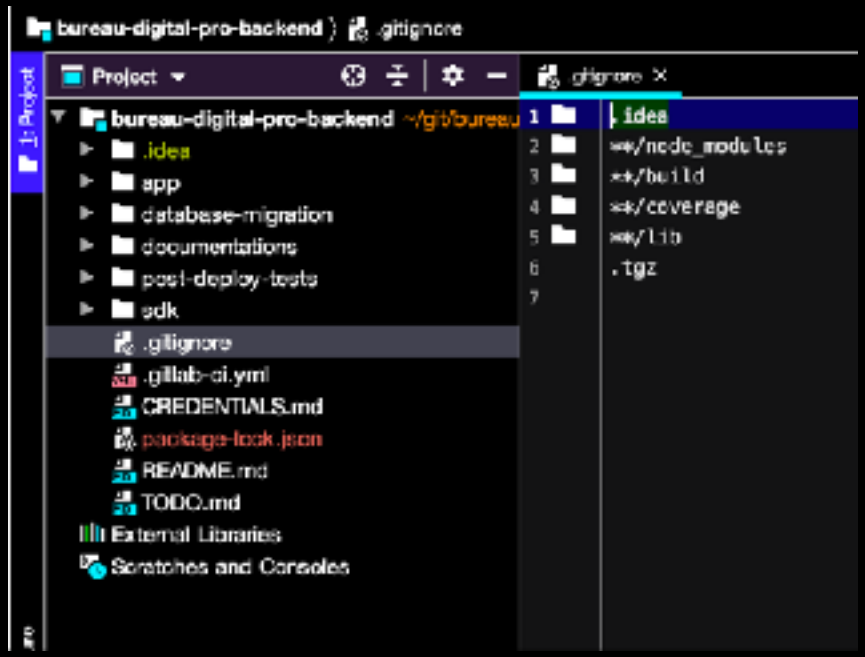
Git stash

Stash : Stocker quelques chose dans un endroit spécifique

- Quand l'utiliser
  - Si tu ne veux pas commit car il te reste du boulot et que tu veux le reprendre plus tard
    - Stash your work (save it) then switch branches
- Comment est ce qu'on Stash
  - Sur la branche actuelle
    - git stash pour sauver le travail (working directory is now clean si vous faite git status)
  - Changer de branche, pour travailler sur quelque chose d'autre
  - Retour sur la branche initiale
    - git stash pop pour récupérer vos modifications

Ignoring files

- Les fichiers que vous ne voulez pas voir dans Git comme untracked
  - Comme les fichiers automatiquement générés (Intellij, log files, build system)
- Le Pattern dans le .gitignore est sous forme de regex :



- Ne pas oublier d’ajouter et de commit .gitignore !

Git additional commands

Tagging

Create Tag
```
$ git tag –a <tagName> –m "msg"
```
List Tags
```
$ git tag
```
Delete Tag
```
$ git tag -d <tagName>
```
Push Tag
```
$ git push <remote>  --tags
$ git push <remote> <tagName>
```

Rebase

Rebase master content into current br
```
$ git rebase master
```
Interactively rebase current branch
```
$ git rebase –i <branch>
```

Stash

Stash modification
```
$ git stash
```
Apply the modification back
```
$ git stash pop
```

Patching

Method 1 (no commit)

Share change from $id1 to $id2
```
$ git diff $id1 $id2 > output.patch
```
To apply
```
$ git apply  output.patch
```

Method 2 (with commit)

Generate patch for last 2 commits
```
$ git format-patch -2
```
Apply patch
```
$ git am  file.patch
```

Resolve merge conflicts

To view merge conflicts
```
$ git diff
```
To discard conflicting patch
```
$ git reset --hard
$ git rebase --skip
```
After resolving conflicts, merge with
```
$ git add [conflict file]
$ git rebase --continue
```

Debugging

Who did what
```
$ git blame
```
Find in source code
```
$ git grep
```
Finding regressions
```
$ git bisect
```

Cherry pick

Apply a change introduced by a commit
```
$ git cherry-pick $id
```

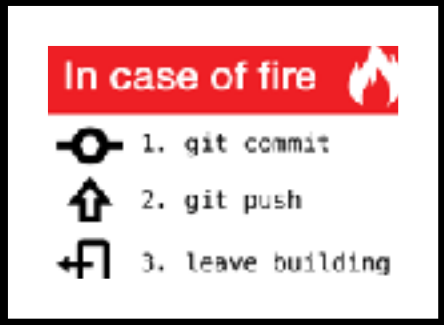
Others

Global Ignore files
```
$ edit .gitignore
```
Configure aliases
```
$ git config --global alias.c "commit"
```
Use git help
```
$ git command --help
```

Git best practices

Commit Often, Perfect Later, Publish Once

- Review code avec git diff avant de commit
- Commits
  - Toujours faire le plus petit commit possible.
  - Le commit message doit être compréhensible
    - En lisant les commit log on doit pouvoir raconter une histoire.
    - Utiliser des templates / projet par exemple :
      - [bug-xxx] fixed bug xxx ou [task-yyy] implementation of task yyy
  - Tester avant de commit !!!



In closing

- Ce n'est que la partie haute de l'iceberg de ce qu'est GIT
- Quelques articles intéressants :
  - What is a version control
  - Why Git
  - A Git workflow for Agile team
  - Git best practices
  - Pro Git (free ebook)
  - Git bisect

- Et bien sûr….
  - git command --help ☺